Advanced Algorithm Design: Semidefinite Programming (SDP)

Lectured by Prof. Moses Charikar
Transcribed by Aaron Schild*

March 25, 2013

In the last few lectures, we considered constant-factor approximation algorithms that relied on linear programming and greedy algorithms. In this lecture, we will analyze algorithms that use a more powerful mathematical programming technique called *semidefinite programming*. We will illustrate the power of semidefinite programming by looking at the maximum cut problem. However, SDPs are useful for other problems as well.

1 The Max-Cut Problem

In the weighted version of the maximum cut problem, we are given a graph $G = (V, E)$ with edge weights given by a function $w : E \to \mathbb{R}_{\geq 0}$. We want to find the set $S \subseteq V$ that maximizes

$$\sum_{e \in E(S, V \setminus S)} w(e)$$

This is superficially similar to the minimum cut problem. Nonetheless, the Max-Cut problem is NP-complete, unlike its counterpart. Furthermore, it is NP-hard to approximate within any ratio better that $\frac{16}{17}$. [1]

*aschild@princeton.edu

2 Some 1/2-approximation algorithms

There are many simple approximation algorithms that obtain a cut containing at least half of the edges of the graph, which is always at least half the weight of the optimal cut. We will discuss two of them.

2.1 A greedy algorithm

Iterate through the vertices once and suppose the interation order is $\{v_1, v_2, \ldots, v_n\}$. While doing so, keep track of a partition of the vertices $S \cup T$ seen so far. Upon seeing v_i, let E_i be the edges with v_i and some vertex v_j with $j < i$ as endpoints. Let E_i^S, E_i^T be a partition of the edges in E_i, where E_i^S is the set of edges with an endpoint in S and E_i^T is the set of edges with an endpoint in T. If $w(E_i^S) > w(E_i^T)$ ($w(F) = \sum_{e \in F} w(e)$ for $F \subseteq E$), then add v_i to T and increment i (which adds E_i^S to the cut). Otherwise, add v_i to S and increment i (which adds E_i^T to the cut).

Note that $\cup_{i=1}^n E_i = E$ and that $E_i \cap E_j = \emptyset$ for $i \neq j$. For each i, at least $\frac{1}{2}w(E_i)$ is added to the cut. Therefore, the total weight of the $S - T$ cut is at least $\frac{1}{2}w(E)$.

2.2 A randomized algorithm

Iterate through the vertices once. When considering a vertex v_i (iteration order as in the previous section), add it to S with probability $1/2$ and add it to T otherwise. The choices for each vertex are made independently.

Now, consider the expected total weight of the resulting $S - T$ cut. Note that $\mathrm{E}[w(E(S,T))] = \sum_{e \in E} \Pr[e \in E(S,T)] w(e) = \sum_{e \in E} \frac{1}{2} w(e) = \frac{1}{2} w(E)$, as desired.

3 Max-Cut with quadratic programming

Are there any more powerful techniques than greed and basic randomness for this problem? Let's start by writing the unweighted max cut problem as a quadratic program. These ideas also apply to the weighted version.

$$\begin{aligned} \text{Maximize} \quad & \sum_{(u,v)\in E} \frac{1 - x_u x_v}{2} \\ \text{subject to} \quad & x_u^2 = 1 \quad \forall u \in V \end{aligned}$$

Figure 1: A quadratic programming formulation of the Max-Cut problem, where $x_v = 1$ and $x_v = -1$ identify membership in S and T respectively

However, quadratic programs are NP-hard to solve (precisely because this is a reduction from max-cut to quadratic programming!) In future sections, we will show that replacing the scalars x_u in this quadratic program with n-dimensional unit vectors results in a polynomial time solvable "relaxation" of the max-cut problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_{(u,v)\in E} \frac{1 - w_u \cdot w_v}{2} \\ \text{subject to} \quad & w_v \cdot w_v = 1 \quad \forall v \in V \\ & w_v \in \mathbb{R}^n \quad \forall v \in V \end{aligned}$$

Figure 2: A vector programming formulation of Max-Cut

By *vector program*, we mean a linear program in terms of variables that are dot products of two vectors.

4 Introduction to SDPs

Recall the standard formulation of a maximization LP (in Figure 3).

An SDP is just an LP in which there are $\frac{n^2+n}{2}$ variables X_{ij} with $X_{ij} = X_{ji}$ instead of n variables. Furthermore, instead of positivity constraints, there is a positive semidefiniteness constraint, where $\{X_{ij}\}$ is viewed as a square matrix in the positive semidefiniteness constraint and a vector in the $A_i X \leq b_i$ constraint (see Figure 4).

$$
\begin{array}{lrl}
\text{Maximize} & & c^T x \\
\text{subject to} & A_i x \leq b_i & 1 \leq i \leq n \\
& x_i \geq 0 & 1 \leq i \leq n
\end{array}
$$

Figure 3: A typical LP formulation

$$
\begin{array}{lrl}
\text{Maximize} & & c^T x \\
\text{subject to} & A_i X \leq b_i & 1 \leq i \leq n \\
& & X \succeq 0
\end{array}
$$

Figure 4: A typical SDP formulation

An $n \times n$ matrix X is *positive semidefinite*, denoted $X \succeq 0$, if and only if any of the following equivalent characterizations holds:

1. X only has nonnegative eigenvalues.
2. For all $y \in \mathbb{R}^n$, $y^T X y \geq 0$.
3. There is some $n \times n$ matrix V such that $X = V^T V$.

By Condition 2, $\alpha X + (1-\alpha)Y$ is positive semidefinite for $\alpha \in (0,1)$ if X and Y are positive semidefinite. Therefore, the space of positive semidefinite matrices, called the *positive semidefinite cone*, is convex. Furthermore, given a point (a matrix X) that is not a solution to the SDP in Figure 4, there is a polynomial time algorithm for producing a separating hyperplane. If X does not satisfy one of the constraints $A_i Z \leq b_i$, then one obtains a separating hyperplane using the same techniques as with linear programming (replace b_i with $b_i + \epsilon$ for some sufficiently small $\epsilon > 0$). What about the positive semidefinite constraint? If X is not positive semidefinite, then by Condition 2 (or rather a polynomial tima algorithm for finding y in Condition 2), we can produce a vector y such that $y^T Z y = \sum_{i,j} Z_{ij} y_i y_j < 0$. Treating the y_is as constants and Z_{ij} as variables gives a hyperplane $\sum_{i,j} Z_{ij} y_i y_j = \frac{1}{2} \sum_{i,j} X_{ij} y_i y_j$. Therefore, we may use the ellipsoid algorithm to produce a solution within ϵ

of the optimal in $poly(1/\epsilon, n)$ time.

Why is it that for SDPs we can obtain approximate solutions while for LPs we can obtain optimal solutions? For LPs, we know that at least one vertex of the LP polytope must be optimal. Therefore, once we get a solution for small enough ϵ with the ellipsoid method, we may "round" the solution to the nearest vertex. With SDPs, the set of possible solutions is not a polytope, so we have no similar guarantees.

5 Applying SDPs to Max-Cut

First, note that Figure 2 is an SDP, as Condition 3 for positive semidefiniteness implies that an SDP is a vector program and vice versa (with $X_{ij} = V_i \cdot V_j$, where $X = V^T V$).

This SDP defines a map from vertices of G to unit vectors in $\mathbb{R}^n$, where $n = |V|$. Now, we will use randomized rounding to convert these unit vectors into a high weight cut. The rounding scheme will partition the unit vectors into two sets by picking a random hyperplane through the origin and designating all vertices corresponding to vectors on one side S. How do we pick such a hyperplane? We pick a random normal vector in $\mathbb{R}^n$ by picking each coordinate independently with a Gaussian distribution.

Now, we will calculate the expected weight of this cut. Since the expectation of a sum is the sum of expectations, we may analyze every edge of G independently. Consider an edge $e = (u, v) \in E$ and consider the intersection of the random hyperplane H with the two dimensional plane generated by w_u and w_v. This intersection will be a line ℓ. ℓ is also a uniformly random line passing through the origin in the plane generated by w_u and w_v. Therefore, the probability that w_u and w_v are on opposite sides of ℓ is $\frac{\theta}{\pi}$, where θ is the angle between w_u and w_v in $[0, \pi]$. Therefore, the expected contribution of e to the weight of the cut is $\frac{\theta}{\pi}$ in the unweighted problem. The contribution of e to the value of the SDP is $\frac{1 - w_u \cdot w_v}{2} = \frac{1 - \cos\theta}{2}$. Therefore, the ratio of the expected cut contribution to the SDP contribution is $\frac{2\theta}{\pi(1-\cos\theta)} \geq .878$ for all $\theta \in [0, \pi]$. Overall, this means that the expected weight of the cut from randomized rounding is at least .878 times the SDP value, which is an upper bound on the optimal maximum cut size. Therefore, this algorithm is

a randomized 0.878-approximation!

Could there be a better approximation algorithm? First, it is unlikely that this SDP can be used to obtain a better algorithm through clever rounding schemes, as the *integrality gap* (ratio of the optimal integer solution to the optimal fractional solution) is 0.878. Could we add more constraints to this SDP to get a better relaxation? One can try adding some triangle inequality constraints of the form

$$(1 - w_p \cdot w_q) + (1 - w_q \cdot w_r) \geq (1 - w_p \cdot w_r)$$

for all triples of vertices $p, q, r \in V$. This is valid because viewing w_p, w_q, w_r as scalars shows that inequalities of this form will be satisfied by any valid cut. One might think that they would help, as not all unit vectors in $\mathbb{R}^n$ satisfy these inequalities. However, it turns out that the integrality gap of the resulting SDP is still .878. All of these results are related to a more fundamental hardness conjecture called the Unique Games Conjecture.

6 The Unique Games Conjecture (UGC)

The Unique Games Conjecture was originally proposed in [2] to serve as a stronger extension of NP-hardness for approximation hardness results. There are many formulations of UGC and we present one involving systems of equations.

Consider variables $x_1, \ldots, x_n \in \mathbb{Z}_p$ and systems of constraints of the form $x_i - x_j \equiv c_{ij} \pmod{p}$ for constants c_{ij}. One can determine if these systems are satisfiable using Gaussian elimination, which takes polynomial time. What if the system cannot be satisfied? Can we figure out what the maximum fraction of constraints is that we can satisfy? UGC, if true, would say that the problem is *very* hard to get reasonable approximations for! More precisely, UGC says that given an $\epsilon > 0$, there is some $p = f(1/\epsilon)$ such that there is no polynomial time algorithm for the following decision problem:

Given: A system of constraints as specified above for which either at least $1 - \epsilon$ or at most ϵ of the constraints can be satisfied.

Decide: Whether there is an assignment that satisfies at least $1-\epsilon$ of the constraints or whether no assignment satisfies more than ϵ of the constraints.

Assuming UGC, one can show that there is no $(.878+\epsilon)$-approximation algorithm for the Max-Cut problem for any constant $\epsilon > 0$! For a proof, see [3].

References

[1] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, July 2001.

[2] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 767–775, New York, NY, USA, 2002. ACM.

[3] Guy Kindler, Ryan O'Donnell, Subhash Khot, and Elchanan Mossel. Optimal inapproximability results for max-cut and other 2-variable csps? *Electronic Colloquium on Computational Complexity (ECCC)*, (101), 2005.